

Merging Logic Programming into Web-based technology: a Coordination-based approach

Enrico Denti, Antonio Natali, Andrea Omicini

Laboratorio di Informatica Avanzata
Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna
Viale Risorgimento, 2 - I-40136 Bologna, Italy

`mailto:{edenti,anatali,aomicini}@deis.unibo.it`

`http://www-lia.deis.unibo.it/Staff/`

Current WWW technology

- “the illusion of several cooperating applications working together”¹
- the de-facto standard (and hardware platform) for groupware applications
- Java as a reference architecture
 - from server-centered to distributed applications
 - event-driven programming model
 - “reactive” components
 - method-invocation-style for interaction
 - continuously evolving towards new abstractions (JavaBeans, Javaspaces...)
- weak with respect to **higher-level middleware services** required by *new applications*
 - highly interactive
 - abstractions to compose entities into new software ensembles
 - effective **coordination capabilities**
 - embedded intelligence

¹ P.Ciancarini, R.Tolksdorf, F.Vitali. *Weaving the Web Using Coordination*. In *Coordination Languages and Models* P. Ciancarini and C. Hankin (Eds), Lecture Notes in Computer Science No. 1061, Springer, 1996, pp. 411-415.

Which role for LP?

- the base to extend the current (Java-based) Web architecture providing *enhanced Linda-like coordination*
- the support to the *integration* between *logic-based* and *conventional applications*
- the support to the concept of *Programmable Coordination Medium*
- the way to provide *inferential capabilities*

Our goal

- to support **interaction** by adopting an **enhanced tuple space** as communication channel, acting:
 - as a **high-level communication device**
(for all components)
 - as a **knowledge base**
(for *logic-based* components only)
- **programmable** so as to embed *the logic of interaction* into the coordination medium
(rather than spreading it over components)
- supporting **inferential activities**:
 - ability to reason over the available knowledge
in a time-dependent environment

As a result:

- a hybrid technology
- extending the current WWW architecture with
a new (tuple space-like) component

Why not just a Tuple Space?

1) a conventional TS **does not capture the logic of interaction** *at the abstraction level required by the application*

Example: Dining Philosophers

- easy to solve if philosophers acquire / release *couples of forks transactionally*
- more complex if philosophers have to handle the fork acquisition protocol *explicitly*
- transaction handling should be moved **from** agents **to** the communication channel
 - possible *if the behaviour of the communication component can be extended*
 - so as to support *the logic of interaction* required by the specific application

2) a conventional TS **does not allow reasonings on communication**

- only state *changes* are visible (in / out)
- rd operations have no observable effect
- it is impossible to trace all kinds of communication events - and reason on them

The ACLT model

- defined in the context of multi-agent systems
- meets most of the previous requirements
- a working system
 - <http://www-lia.deis.unibo.it/Software/ACLT>
- focused on the model of *component interaction*
 - what is relevant is the *observable behaviour* of (*heterogeneous*) components (agents)
 - promotes system architectures designed *as independently as possible* of the internal models of single components
- **Linda-like**, *multiple*, *logic* tuple spaces
- *programmable* through *reactions*
- able to support *logic inferences* (*demos*)
 - **twofold interpretation** of tuple spaces
 - *knowledge bases* for logic components
 - *simple communication devices* for all (logic and non-logic) components
 - **knowledge classification**
 - *persistent knowledge* cannot be withdrawn
 - *transient knowledge* cannot be used in *demos*

Programmable Tuple Spaces

- all communication events are made *observable*
 - observability is enhanced from *tuples* to *operations on tuples*
 - out corresponds to a single event
 - in / rd correspond to two events
(*query phase* / *answer phase*)
 - all the *properties* of the occurred communication event are available (*current tuple*, *current operation*, *current agent*, ...)
- communication events are mapped onto *logical events*
- “*scripts*” (*reactions*) are *event handlers* associated to logical events’ occurrence
 - they can *access all the properties* of the occurred communication event
 - and also *the whole current communication state* available in the tuple space
- so, they provide a flexible way to *extend* (and *specialise*) the *observable Tuple Space behaviour*

The Reaction Model

- *multiple reactions* may be triggered after a *single* communication event
 - each reaction constitutes a separate *transaction*
 - **reactions do not alter the intended semantics** of the communication operation *at the system level*
 - they perform *further tasks*
 - executed *after* the triggering event
 - reactions triggered by a communication event
 - are executed in a *non-deterministic order*
 - *cannot be nested*
 - can trigger other (*indirect*) reactions
 - but all will be completed *before* serving the next communication event
 - **so, reactions CAN alter the intended semantics** of the communication operation *at the agent level*
 - *agents perceive only the final result of*
 - the original communication event
 - all (*direct* and *indirect*) reactions
- as a single transition** of the tuple space state

ACLT Reaction Syntax

- a reaction is specified through tuples which are unitary clauses

- to map *communication events* to *logical events*

`map(Operation, LogicalEvent)`

- to associate *reactions* to *logical events*

`react(LogicalEvent, ReactionBody)`

- if the mapping is 1-1, `map + react = reaction`

`reaction(Operation, ReactionBody)`

Reaction specification language

- communication primitives (`in_r/1`, `rd_r/1`, `out_r/1`)
- state primitives (`current_tuple/1`, `current_op/1`, ...)
- term predicates (equality, unifiability, ...)

```
reaction(out(forks(F1,F2)), (  
  in_r(forks(F1,F2)),  
  out_r(fork(F1)), out_r(fork(F2)) ))
```

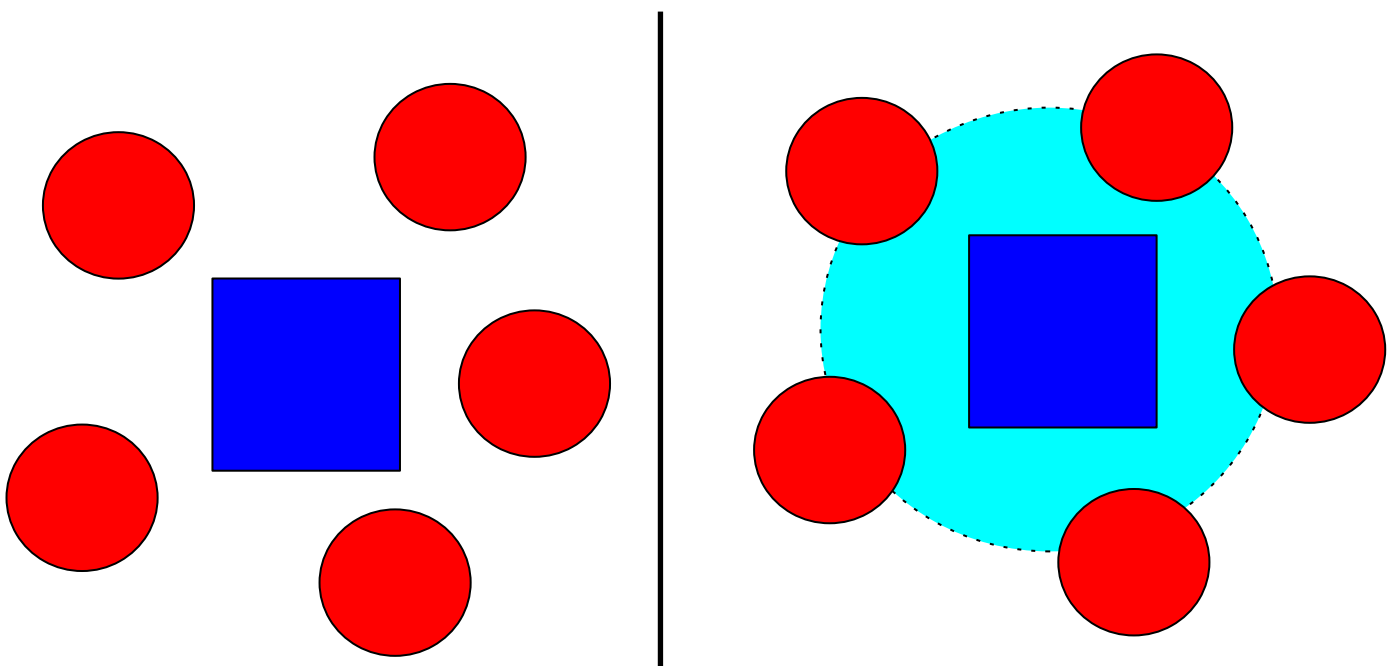
- primitives inside *ReactionBody* are executed *sequentially*
- but their side effects will be *actualised only if the transaction commits*
 - if something fails, reaction aborts
→ as if it never happened

The new Tuple Space role

- more than just a communication channel
- a *ready-to-extend* system resource
- **programmable** so as to support the coordination policies required by the problem
- freeing agents from explicitly handling low-level interaction details

Changing design approach

- **FROM** *agents explicitly charged* of handling a (possibly complex) interaction protocol
- **TO** agents designed according to a (possibly) straightforward communication protocol **AND** *the coordination medium charged* of (most of) the coordination tasks



Logic tuple spaces

Supporting logic inferences

- for logic components only
- **which notion of logic consequence** in such a time-dependent environment ?
 - platonic view of immutable truth (typical of logic)
vs. transient nature of knowledge in tuple spaces
 - correctness & completeness ?

ACLT “demo” model

- **correctness** → **knowledge classification**
 - **transient** vs **persistent** knowledge
 - logic proofs on persistent knowledge only
- **completeness** → keep new knowledge into account (on backtracking)
- **synchronisation** (is it “*too early to deduce*” ?)

ACLT hybrid “demo” primitives

	<i>unsynchronised</i>	<i>synchronised</i>
<i>incomplete</i>	demo	demoWait
<i>complete</i>	demoLast	demoWaitLast

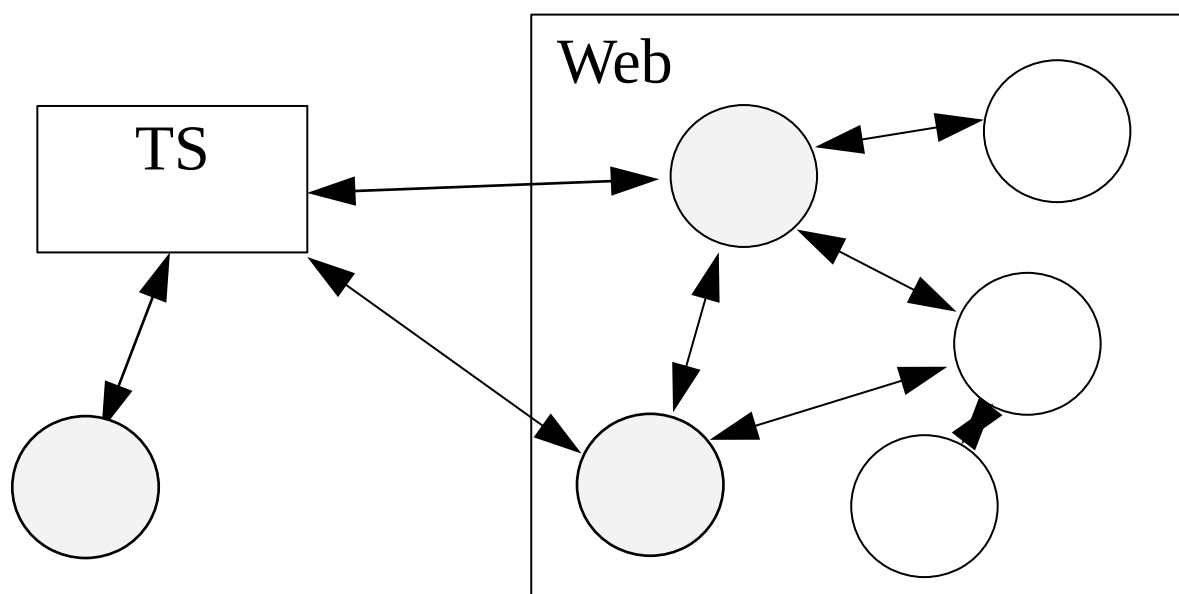
Moving to the Web platform

Basic ideas

- adding a Programmable, Logic Tuple Space to the current Web architecture
 - ACLT as a reference model
 - ACLT as an existing (working) system

Advantages

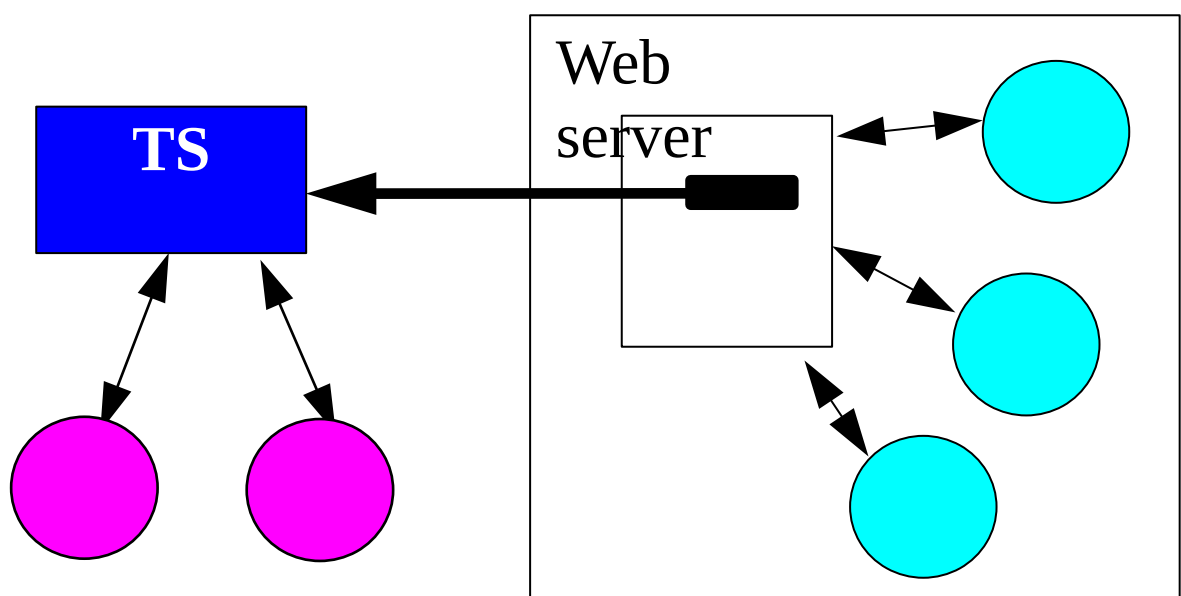
- overcoming point-to-point communication
- easily adding intelligent (heterogeneous) components to the system
 - *low impact on the overall system architecture*
 - but (at least some) components *must be aware* of the logic TS and *exploit it* to store knowledge
- software layer needed to interact might be downloaded from the net when needed (plug-in, Java Bean,...)



One first scenario

A “conservative” approach

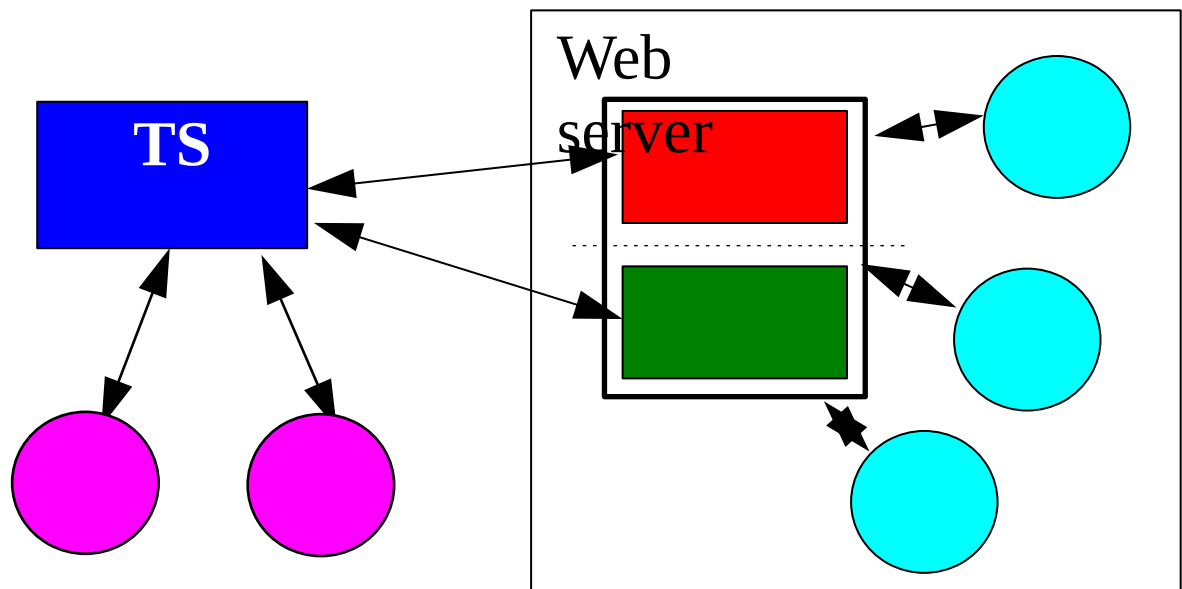
- **purpose:**
 - allow **logic inferences** on the Web server activity
- **requirement:**
 - **HTTP events must be made *observable***
- **consequence:**
 - the Web server must be slightly ***extended***
 - but the change is like adding a `printf()`
 - the HTTP interaction protocol remains *untouched*
- **what for?**
 - tracing, statistics, monitoring purposes
- **limit:**
 - knowledge in the TS (and subsequent reasonings) **cannot influence the Web server behaviour**



Another scenario

A more “innovative” approach

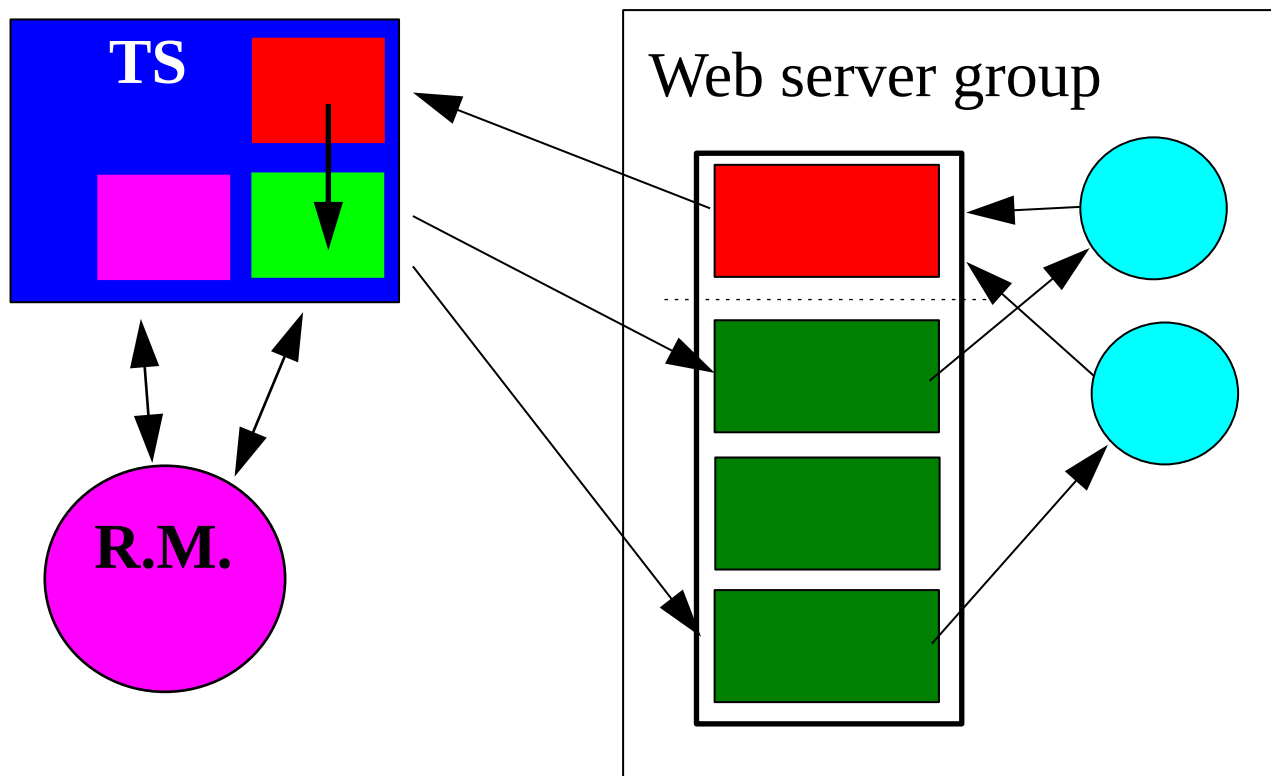
- allowing the new component *influence (and change) the Web server behaviour* based on the knowledge stored in the TS
- the Web server architecture has to be changed
 - a simple “event mirroring” is no longer enough
 - communication must occur through the TS
 - though nothing should change externally (HTTP)
- *uncoupling the two components of the Web server* (receiver & transmitter)
- *only one coordination protocol* (+ HTTP for usual client/server communication)



A sample application

The resource manager

- A logic agent handling a Web server group
 - reasoning on the knowledge in the TS
- **Reasoning on what?**
 - • “raw” HTTP events ...?
 - • *higher-level events* (as observable logic tuples)
 - synthesised through *reactions*
- **Relevant HTTP event:**
 - the request of a new connection
- As a result:
 - Server behaviour depending on



Conclusions

- The Web as a software platform demands
 - higher-level communication abstractions
 - coordination capabilities
 - ability to support heterogeneous agents
- Logic Programming can *not* radically change Web systems
 - but can provide *inferential capabilities*
 - possibly through *logic components*
- The ACLT model / system provides *logic*, *programmable* tuple spaces
 - twofold TS interpretation (logic theory / TS)
 - *demo* primitives for safe deductions (+ synchr.)
 - *reactions* to make Tuple Spaces *programmable*

Our challenge

- to provide a tool allowing Prolog agents to run together / interact with Java agents
 - let a larger community perceive LP advantages
- facing the price to be paid
 - adapt LP model to the notion of modifiable theory
 - perhaps the most critical issue to merge LP with Internet technology ... ?

Related work

- **Linda Interactor (P.Tarau, K.De Bosschere)**
 - heterogeneous - BinProlog & Java - agents
 - no *demos* - backtrackable in instead
 - no reaction programmability
- **JProlog (Tarau et al.)**
 - Java as a bytecode for Prolog
- **Javaspaces** (chatsubo.javasoft.com/javaspaces)
 - rich typing (and subtyping) vs. plain clauses
 - Java objects as tuple fields (with methods)
 - *notification* instead of blocking *in* and *rd*
- **PiLLoW / CIAO library (M. Hermenegildo et al.)**
 - LP/CLP used for Internet/WWW programming
 - LP scripts for CGI apps & HTML form handling
 - *active modules* (providing “remote relations”)
 - local execution of downloadable Prolog modules (via a local “helper application” Prolog engine)
 - different goals - no tuple spaces
- **Shared Prolog / Polis (P.Ciancarini)**
 - similar notion of “reactive agents”
 - no application agents, no *demos*
- **Gamma** (Banatre, Le Métayer)
 - somehow similar notion of reaction
 - a pure abstract model - intentionally!

References

Coordination, Linda & The Internet

- P.Ciancarini, R.Tolksdorf, F.Vitali. *Weaving the Web Using Coordination*.
In *Coordination Languages and Models*, P. Ciancarini and C. Hankin (Eds),
Lecture Notes in Computer Science No. 1061, Springer Verlag, Germany, 1996
- P.Ciancarini. *Coordination Models as Software Integrators*. ACM Computing
Surveys, 28(2), June 1996.
- K. De Bosschere, P. Tarau. *Blackboard-based Extensions in Prolog*. Software
Practice and Experience, 26(1), January 1996, pp. 49-69, John Wiley & Sons,
Ltd.
- K. De Bosschere, P. Tarau. *Virtual World Brokerage with BinProlog and Netscape*.
<http://clement.info.umoncton.ca/~lpnet/lp-internet/lpnet10/art.html>
- P. Tarau, V.Dahl, K. De Bosschere. *High Level Logic Programming Tools for
Remote Execution, Mobile Code and Agents*.
<http://clement.info.umoncton.ca/html/remote/art.html>
- P.Ciancarini. *Distributed Programming with Logic Tuple Spaces*. New Generation
Computing, 12, 1994.

ACLT Coordination model

- A.Omicini, E.Denti, A.Natali. *Agent Communication and Control through Logic
Theories*. In *Topics in Artificial Intelligence*, M. Gori and G. Soda (Eds),
Lecture Notes in Artificial Intelligence No. 992, Springer, Germany, 1995
- E.Denti, A.Natali, A.Omicini, M.Venuti. *Logic Tuple Spaces for the Coordination of
Heterogeneous Agents*. In *Frontiers of Combining Systems*, F. Baader and
K. U. Schulz (Eds.), Kluwer, USA, 1996
- E.Denti, A.Natali, A.Omicini, M.Venuti. *An Extensible Framework for the
Development of Coordinated Applications*. In *Coordination Languages and
Models* P. Ciancarini and C. Hankin (Eds), LNCS No. 1061, Springer,
Germany, 1996.
- E.Denti, A.Omicini. *Designing Multi-Agent Systems around an Extensible
Communication Abstraction*. 4th ModelAge workshop on Formal Models of
Agents. A. Cesta and P.Y. Schoebbens (Eds), Ist. Psicologia CNR, Rome, Italy,
1997. (To appear also in the the LNAI series by, Springer Verlag - 2nd
quarter 1997)

- E.Denti, A.Natali, A.Omicini. *Programmable Coordination Media*. Proceedings of Coordination '97 (Berlin, Germany, 1-3 September 1997). Lecture Notes in Computer Science, Springer Verlag, Germany, 1997. To appear.
- F.Zanichelli, S.Caselli, A.Natali, A.Omicini. *A Multi-Agent Framework and Programming Environment for Autonomous Robotics*. Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'94, San Diego (CA), USA, May 11-13, 1994. IEEE Computer Society Press, pp. 3501-3506.